IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	Nir N. Shavit et al.		
Title:	MAINTAINING A DOUBLE-ENDED QUEUE IN A CONTIGUOUS ARRAY WITH CONCURRENT NON-BLOCKING INSERT AND REMOVE OPERATIONS USING A DOUBLE COMPARE-AND-SWAP PRIMITIVE		
Application No.:	09/547,288	Filed:	April 11, 2000
Examiner:	Aimee J. Li	Group Art Unit:	2183
Atty. Docket No.:	004-4663	Confirmation No.:	4871

November 7, 2005

Mail Stop Appeal Briefs - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF (37 C.F.R. § 41.37)

This brief is in furtherance of the Notice of Appeal, filed on May 6, 2005. The fee required under 37 C.F.R. § 41.20(b)(2) is provided in the accompanying Transmittal.

REAL PARTY IN INTEREST

The real party in interest in this appeal is Sun Microsystems, Inc., the assignee of record, as evidenced by the assignment recorded at Reel/Frame 10712/617.

RELATED APPEALS AND INTERFERENCES

Appellant has no knowledge of any related appeals or interferences.

STATUS OF CLAIMS

Claims 1 – 43 are pending. Claims 1 – 43 stand as rejected. Claims 1 – 43, now presented herein on appeal, are reproduced in the Appendix attached hereto.

STATUS OF AMENDMENTS

No amendments have been filed subsequent to the final rejection.

SUMMARY OF CLAIMED SUBJECT MATTER

As a general matter, systems, articles and/or methods in accordance with the presently claimed subject matter can be employed in software objects, structures and mechanisms suitable for coordinated concurrent operations on data structures (e.g., by processes, threads or other sequences of instructions). Atomic, dual-target compare-and-swaps (DCASs) or, more-generally, atomic multi-way compare-and-swaps are employed in specific ways (recited in the claims) to coordinate access to data structures even in the presence of concurrent operations thereon. In addition to the specific details of DCAS (or multi-way compare-and-swap) utilization to update/access particular targets in such data structures, each of the claims recite particular utilizations of the DCAS (or multi-way compare-and-swap) with respect to detection (or disambiguation) of boundary condition states.

For example, some realizations of the presently claimed subject matter provide for a set of structures and techniques for implementing an object susceptible to concurrent access with an atomic dual target compare and swap operation, which returns indication of a boundary condition state, e.g., full or empty. *See* page 3, paragraph [1008] – [1009]. One characteristic of some of these implementations is that a processor can detect these boundary cases, e.g., determine whether the array is empty or full, without checking the relative locations of the two end pointers in an atomic operation. *See* page 3, paragraph [1009].

Independent claim 1 is directed to a method of managing access to an array susceptible to concurrent operations on a sequence encoded in the array. The method includes executing, as part of a pop operation, an atomic dual target compare and swap (DCAS) operation to update specific targets, namely a then-current, end identifying index for the array and an element of the array adjacent to that identified by the end identifying index. On failure of the DCAS, an indication is returned by which an empty state for the array is detectable. *See* pages 10 – 13, paragraphs [1031] - [1040], and page 15 paragraph [1046]; Figures 1A – 1B, and 4.

Independent claim 6 is directed to executing a DCAS as part of a push operation. As with claim 1, the method includes executing, as part of the push operation, an atomic dual target compare and swap (DCAS) operation to update specific targets. In this case, the specific targets are a then-current, end identifying index for the array and an element of the array identified by the end identifying index. On failure of the DCAS operation that is part of the push operation, an indication is returned by which a full state for the array is detectable. *See* pages 13 – 15, paragraphs [1041] – [1046], and page 15, paragraph [1047]; Figures 3 and 5A – 5C.

Independent claim 10 is directed to a method of providing concurrent access to a double-ended data structure of bounded size implemented using a circular buffer technique. The method includes performing in alternate legs of a conditional branch a first atomic multi-way compare and swap operation and a second atomic multi-way compare and swap operation. The first atomic multi-way compare and swap operation disambiguates a retry state and a boundary condition state of the double-ended data structure. Upon failure of the second atomic multi-way compare and swap operation, the second atomic multi-way compare and swap operation returns an indication disambiguating a retry state and the boundary condition state of the double-ended data structure. *See* page 11 paragraph [1035] and page 13, paragraph [1041].

Independent claim 15 is directed to a method of managing concurrent access to double-ended queue (dequeue). The method comprises executing a DCAS operation in an implementation of a pop operation to interrogate instantaneous values of a first end index and a dequeue element adjacent to the element identified by the first end index for a signature indicative of an empty state. *See* pages 10 – 13, paragraphs [1033] – [1040] and page 15, paragraph [1046].

Independent claim 23 is similar to claim 15, except that it is directed to an implementation of a push operation and a signature indicative of a full state. *See* pages 13 – 15, paragraphs [1041] – [1045] and page 15, paragraph [1047].

Independent claim 25 is directed to executing competing first and second access operations on an array. As part of executing each of the access operations, a DCAS operation is executed to update an end-identifying index and a corresponding element of the array. If successful completion of one of the competing access operations results in a boundary condition

for the array, then the DCAS operation of the other access operation returns an indication of the boundary condition. *See* pages 9 – 15, paragraphs [1031] – [1047]; Figures 1A – 1B, 2, 3, 4, and 5A – 5C.

Independent claim 32 is directed to a double-ended queue (dequeue) implementation comprising a left index L and a right index R into a contiguous array S. The indices L and R together define a circular buffer with state. The entries S[L] and S[R] encode a distinguishing value. The double-ended queue also comprises a computer readable encoding of a first access operation operating at a particular end of the of the sequence and employing a DCAS to update one but not both of the indices L and R, as well as an element adjacent to the element identified by the one index. *See* pages 8 – 15, paragraphs [1026] – [1047]; Figures 5A – 5C.

Independent claim 36 is directed to a concurrent shared object implementation that comprises a push operation and a pop operation. Each of the operations employs a DCAS. Upon failure of the push operation's DCAS operation, the DCAS operation returns an indication by which a full state for the object is detectable. Upon failure of the pop operation's DCAS operation, the DCAS operation returns an indication by which an empty state of the object is detectable. *See* pages 9 – 15, paragraphs [1031] – [1047]; Figures 1A – 1B, 2, 3, 4, and 5A – 5C.

Independent claim 40 is directed to a computer program product encoded in at least one computer readable medium, the computer program product comprising at least one functional sequence implementing an access operation on a concurrent shared object. Instances of the functional sequence are concurrently executable by plural processors of a multiprocessor, and each of the instances includes a DCAS. The DCAS operation of an instance of a functional sequence is responsive to a corresponding boundary condition state of the concurrent shared object. *See* pages 8 – 15, paragraphs [1026] – [1047].

Independent claim 43 is directed to an apparatus comprising means for coordinating competing access operations. The coordinating means include processor(s) executing code that employs a DCAS operation in each instance of the competing access operations to disambiguate a retry state and a boundary condition state of the array. *See* pages 5 – 6, paragraphs [1019] - [1021], pages 9 – 15, paragraphs [1031] – [1047].

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Ground I: The rejection of claims 1, 2, 4-9, 25-31, and 36-39 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 4,584,640 issued to MacGregor in view of U.S. Patent No. 6,128,710 issued to Greenspan et al., and in view of U.S. Patent No. 5,081,572 issued to Arnold.

Ground II: The rejection of claims 10-16 and 20-24 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,081,572 issued to Arnold in view of U.S. Patent No. 6,128,710 issued to Greenspan et al.

Ground III: The rejection of claims 32, 35, 40 and 42 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 4,584,640 issued to MacGregor in view of U.S. Patent No. 6,128,710 issued to Greenspan et al., and in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999.

Ground IV: The rejection of claims 33-34 and 41 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 4,584,640 issued to MacGregor in view of U.S. Patent No. 6,128,710 issued to Greenspan et al. and in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999as applied to claim 32, and further in view of U.S. Patent No. 5,081,572 issued to Arnold.

Ground V: The rejection of claim 43 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,081,572 issued to Arnold in view of U.S. Patent No. 6,128,710 issued to Greenspan et al., and in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999.

ARGUMENT

As a general proposition, obviousness is a legal determination based on underlying factual inquiries. *See* Minnesota Min. & Mfg. Co. v. Johnson & Johnson Orthopedics, Inc., 976 F.2d 1559, 24 U.S.P.Q.2d (BNA) 1321, 1332-1333 (Fed. Cir. 1992). Graham v. John Deere Co., 383 U.S. 1, 17 (1966) defines the factual inquiries utilized to evaluate the prior art. Specifically, the prior art is evaluated in terms of: (1) its scope and content; (2) the differences between the

prior art and the claimed invention; (3) the level of ordinary skill in the art at the time the application was filed; and (4) objective, or secondary, evidence of nonobviousness such as commercial success, failure of others, long-felt need and unexpected results, which must be considered in reaching a conclusion of obviousness. Graham v. John Deere Co., 383 U.S. 1, 17, 148 U.S.P.Q. 459, 460 (1966); Panduit Corp. v. Dennison Mfg. Co., 810 F.2d 1561, 1566-67, 1 U.S.P.Q.2d 1593, 1595-96 (Fed. Cir. 1987); Minnesota Min. & Mfg. Co. v. Johnson & Johnson Orthopaedics, Inc., 976 F.2d 1559, 24 U.S.P.Q.2d 1321, 1333 (Fed. Cir. 1992).

In the present appeal, the issue relates to specific differences between the prior art and appealed claims. All claim limitations must be considered in the obviousness analysis. Indeed, it is clear error to ignore limitations clearly set forth in the claims. *See* Panduit Corp., 1 U.S.P.Q.2d at 1603–04, 810 F.2d at 1576. Here the Office has ignored clear limitations of the claims.

In rejecting claims under 35 U.S.C. § 103, the examiner bears the initial burden of presenting a *prima facie* case of obviousness. *See* In re Rijckaert, 9 F.3d 1531, 1532, 28 U.S.P.Q.2d (BNA) 1955, 1956 (Fed. Cir. 1993). A *prima facie* case of obviousness is established by presenting evidence that would have led one of ordinary skill in the art to combine the relevant teachings of the references to arrive at the claimed invention. *See* In re Fine, 837 F.2d 1071, 1074, 5 U.S.P.Q.2d (BNA) 1596, 1598 (Fed. Cir. 1988), In re Lintner, 458 F.2d 1013, 1016, 173 U.S.P.Q. (BNA) 560, 562 (CCPA 1972).

CONTENTIONS WITH RESPECT TO GROUND I

Claims 1, 2, 4-9, 25-31, and 36-39 were rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 4,584,640 to MacGregor (hereinafter "*MacGregor*") in view of U.S. Patent No. 6,128,710 to Greenspan et al (hereinafter "*Greenspan*"), and in view of U.S. Patent No. 5,081,572 to Arnold (hereinafter "*Arnold*"). In maintaining the present rejection(s), the Office:

1) Asserts an interpretation of *MacGregor* that does not comport with the disclosure thereof;

   In particular, MacGregor's disclosure is of operations on a doubly-linked list, not on an array as recited in applicant's claims. This is significant not only because

of the structural differences between doubly-linked lists and arrays, but also because the rejected claims variously recite specific pairs of targets for applicant's DCAS uses. Those recited pairs are simply not consistent with the targets of any similar operation in MacGregor's disclosure.

In short, the Office's theory is apparently that mere use of a DCAS-type operation in the context of a push- or pop-type operation disposes Applicant's claim limitations. Such reasoning constitutes legal error in that it ignores specific limitations of the claims.

2) Asserts an interpretation of *Arnold* that does not comport with the disclosure thereof; and

In particular, the Office relies on Arnold for specific claim limitations that relate to applicant's particular usage of a DCAS, namely that the DCAS is employed in a way that "return[s] ... on failure [of the DCAS], an indication by which an empty state of the array is detectable." While Arnold does (i) check for an empty state and (ii) employ a DCAS-type operation, there required relation is absent from Arnold. Indeed, Arnold's empty check is performed before it's DCAS-type operation and no return indication from the DCAS-type operation is employed.

The Office's theory is apparently that (i) Arnold checks for an empty state, that (ii) Arnold employs a DCAS-type operation and that (i) + (ii) good enough. With all due respect, Applicant's claim is a bit more precise in its recitation of the particular source of the relevant indication.

Accordingly, the Office's rejections are based on analyses of relied upon references that attribute to those references disclosure that is simply not contained therein. Furthermore, the Office's rejections are based on a piecemeal aggregation of items improperly teased from the relied upon disclosure. In either case, the Office has no met its burden in making out a *prima facie* case of obviousness and the rejections must be reversed.

We now turn with specificity to the rejected claims.

Independent claims 1 and 6

Though of substantially differing scope, claims 1 and 6 each recite:

1) methods of managing access to **an array;**

2) execution, as part of an pop (or push) operation, of an **atomic dual target compare and swap (DCAS) to update** *a pair of specific recited targets, namely*:

- a then-current, **end-identifying index for the array** AND

- an **element** of the array. *In the case of the pop operation, the element is recited as* an **element of the array adjacent to that identified by the end identifying index.** *In the case of the push operation, the element is recited as* an **element of the array identified by the end identifying index.**

3) returning from the DCAS, on failure thereof, an indication by which a *particular state* of the array is detectable. *In the case of the pop operation, the particular state is* an **empty state of the array.** *In the case of the push operation, the particular state is* a **full state of the array.**

Relied upon Art does not Disclose or Suggest the Specific DCAS Targets Recited

Turning to Office's rejections, *MacGregor's* disclosure is of operations on a doubly-linked list, not on an array as recited in applicant's claims. This is significant not only because of the structural differences between doubly-linked lists and arrays, but also because the rejected claims recite specific pairs of targets for applicant's DCAS uses. In claim 1, i.e., for the pop operation, the recited pair is:

- a then-current, **end-identifying index for the array** AND
- an **element of the array adjacent to that identified by the end-identifying index.**

In claim 6, i.e., for the push operation, the recited pair is:

- a then-current, **end-identifying index for the array** AND
- an **element of the array identified by the end-identifying index.**

Those recited pairs are simply not consistent with the targets of any similar operation in MacGregor's disclosure.[1] Indeed, *MacGregor's* operations, which manipulate a doubly linked list, target a pair of *pointers* (a tail pointer and an internal list *pointer*), not an index and *element* of the array adjacent to that identified by the index or (in the case of the push operation) an index

[1] While the Office states: "MacGregor has not taught atomic compare and swap" and instead relies upon *Greenspan*, Final Action at ¶ 2, that interpretation of *MacGregor* is not entirely correct. *MacGregor* does disclose use (in a deletion operation) of a CAS2 targeting a "tail pointer" and the "next pointer" of a 2nd to last node of the list. The Office apparently ignores both *MacGregor's* CAS2 usage and the actual targets of such such usage in *MacGregor's* doubly-linked list. Those targets are of course different than targets recited in Applicant's claims. *Greenspan* does not add to the Office's position.

and *element* of the array identified by the index. While a tail pointer (in *MacGregor's* list) and a tail-end identifying index (in Applicant's array) may fill similar roles, the same cannot be said of the second targets. *MacGregor's* internal list pointer and Applicant's array element are not identical or even similar in there roles in the respective data structures. In particular, *MacGregor's* internal list pointer identifies another node of the list. Applicant's array element does not. Neither *Arnold* nor *Greenspan* add the missing limitations.

Thus, to maintain it's rejections of claims 1 and 6, the Office must ignore both Applicant's recitation of an *array* access technique (rather than a list access technique) and Applicant's specific language as to the respective second targets of the recited DCAS operations. The Office has ignored specific limitations of the claims. Accordingly, it has committed legal error and, for at least these reasons, **the rejections of claims 1 and 6 (as well as those that depend therefrom) should be reversed.**

Relied upon Art does not Disclose or Suggest the Recited Usage of Indications Returned from the DCAS Operations

The Office further compounds its errors of interpretation by (in effect) ignoring additional language of claims 1 and 6, which recites specific limitations as to the indications returned from the previously-described DCAS operations. In particular, claim 1 recites (with respect to the pop usage) returning an indication by which an *empty state* of the array is detectable. Claim 6 recites (with respect to the push usage) returning an indication by which a *full state* of the array is detectable.

With respect to claim 1, the Office relies on *Arnold's* disclosure of a DEQUEUE operation. In particular, *Arnold* depicts (in Figure 4A, and in code of Table 2) a determination of whether his queue is empty. *That determination is simply not performed using any indication returned from any DCAS, let alone failure of a particular DCAS as claimed.* Indeed, it is clear and unambiguous that such a determination (*see* Arnold, Fig. 4A, element 232, and related description) is performed *prior to* execution of a compare and swap disjoint (CSD), which follows in the flow (Fig. 4B, element 240). While Applicant's do not concede that *Arnold's* CSD usage even corresponds to the recited usage of a DCAS targeted as described above, even *assuming arguendo* some correspondence, it is clear and unambiguous that no indication

returned from *Arnold's* CSD is used in any way to detect an empty state. Of course, neither *MacGregor* nor *Greenspan* provides the missing disclosure. **For this reason alone, claim 1 is allowable and Applicant requests that this Honorable Board reverse the present rejection.**

With respect to claim 6, the Office apparently relies on *Arnold's* disclosure of an ENQUEUE operation (although the Examiner also includes citations for DEQUEUE and pop from stack operations). While the Office asserts that *Arnold* has taught returning from a DCAS, on failure thereof, an indication by which a *full state* of the array is detectable, a review of the actual disclosure of *Arnold*, indicates otherwise. Indeed, Arnold again fails to disclose Applicant's limitation for the reasons described above with regard to claim 1. As before, determination of queue state (Fig. 6, element 102) is performed *prior to* execution of a compare and swap disjoint (CSD), which follows in the flow (Fig. 6, element 106). As before, it is clear and unambiguous that no indication returned from *Arnold's* CSD is used in any way to detect the queue state.

However, and perhaps more significantly, whatever queue state check the Office is relying upon, it is for "empty state," not *full state* of the array as recited in claim 6. To a person of skill in the art, it would not be surprising that *Arnold* is uninterested in a full state, since like *MacGregor*, *Arnold's* disclosure concerns manipulation of a list data structure (which is unbounded in size), not an array (as recited in applicant's claims) in which (at least for a fixed sizing) full state is significant. As before, the Office misinterprets *Arnold* and there is simply no basis for the assertion that Arnold teaches or suggests Applicant's claim limitations related to return of an indication by which full state is detectable. **For this reason as well, claim 6 is allowable and Applicant requests that this Honorable Board reverse the present rejection.**

Independent claim 25

The scope of claim 25 differs substantial from that of claims 1 and 6. Nonetheless, several errors in the rejection of claim 25 (and those dependent therefrom) mimic those described above with regard to claims 1 and 6. In particular, claim recites:

1) a method of managing concurrent **access to an array**;

2) execution, as part of competing first and second access operations, of respective atomic dual target compare and swaps (DCAS) to update respective pairs of

specific recited targets. As before, the targets include respective then-current, end-identifying indices AND **respective elements of the array** corresponding to the respective indices.

3) boundary condition state **indication based on the return from a failing one of the DCAS operations**.

As before, neither *MacGregor*, nor *Arnold*, nor *Greenspan* disclose or suggest these limitations of invention(s) as claimed. Accordingly, claim 25 and those dependent therefrom are all allowable for at least these reasons and a notice to that effect is respectfully requested.

Dependent claim 29

Additionally, with respect to dependent claim 29, *MacGregor* and *Arnold* each fail to disclose or suggest, taken alone or in combination, full state detection in an array. Indeed, as before, absence of such disclosure is not surprising as *MacGregor* and *Arnold* concern manipulation of list data structures (unbounded in size), not arrays (as recited in applicant's claims) in which (at least for a current fixed sizing) full state is significant. *Greenspan* does not alter the conclusion. Claim 29 and those dependent therefrom are allowable for these reasons as well.

Independent claim 36

The scope of claim 36 differs substantial from that of claims 1, 6 and 25. Nonetheless, several errors in the rejection of claim 36 (and those dependent therefrom) mimic those described above with regard to claims 1, 6 and 25. In particular, claim recites a concurrent shared object implementation including:

1) **an array** (more particularly, a contiguous array);

2) computer readable encodings of push and pop operations that employ respective atomic dual target compare and swaps (DCAS) to update respective pairs of specific recited targets. The targets include respective indices AND **respective elements of the array** corresponding to the respective indices.

3) boundary condition state **indication based on the return from a failing one of the DCAS operations**. In particular, a *full state* indication on failure of the DCAS of the push operation and an *empty state* indication on failure of the DCAS of the pop operation.

As before, neither *MacGregor*, nor *Arnold*, nor *Greenspan* disclose or suggest these limitations of invention(s) as claimed. Accordingly, claim 36 and those dependent therefrom are all allowable for at least these reasons and a notice to that effect is respectfully requested.

CONTENTIONS WITH RESPECT TO GROUND II REJECTIONS

Claims 10-16 and 20-24 have been finally rejected under 35 U.S.C. § 103(a) as being unpatentable over *Arnold* in view of *Greenspan*. The Office relies on *Arnold* for the basic limitations, turning to *Greenspan* for an atomic compare-and-swap. As before, the rejections are based on improper interpretation of the relied upon references and, as such, fail to identify teachings or suggestions corresponding to various specifically recited limitations of the rejected claims. As a result, the Office has failed to make out a *prima facie* case of obviousness.

We now turn to the language of specific claims.

Independent Claim 10

Claim 10 recites a method of providing access to a double-ended data structure. The method specifically requires first and second atomic multi-way compare and swaps in alternate legs of a conditional branch. The claim recites that the conditional branch discriminates between presence and absence of a distinguishing value in an element of the data structure corresponding to contents of an index store. Furthermore, specific limitations are recited for respective targets of the first and second multi-way compare and swaps and for particular uses in disambiguating states. For convenience, the claim is duplicated below:

```
10. A method of providing concurrent access to a double-ended
data structure of bounded size implemented using a circular buffer
technique, the method comprising:
     as part of an access to a first-end of the double-ended data
          structure, performing in alternate legs of a conditional
          branch:
          a first atomic multi-way compare and swap on then-current
               contents of a first-end index store and a
               corresponding element of the double-ended data
               structure to disambiguate a retry state and a boundary
               condition state of the double-ended data structure;
          a second atomic multi-way compare and swap on then-current
               contents of the first-end index store and a
               corresponding element of the double-ended data
```

```
            structure, the second multi-way compare and swap
            performing the access and, on failure thereof,
            returning an indication disambiguating a retry state
            and the boundary condition state of the double-ended
            data structure,
    wherein the conditional branch discriminates between presence and
    absence of a distinguishing value in an element of the double-
    ended data structure corresponding to the then-current contents
    of the first-end index store.
```

As a preliminary matter, the relied upon portion of *Arnold* simply does not disclose or suggest respective first and second multi-way compare and swaps in alternate legs of a conditional branch. With all due deference, the Office's assertion in this regard is specious and without any basis in fact. Of course, of *Greenspan* does not add the missing disclosure. For this reason alone, claim 10 (and those dependent therefrom) are allowable and a notice to that effect is respectfully requested.

Further, though the scope of claim 36 differs substantially from that of claims 1, 6, 25 and 36, several interpretational errors mimic those described above. In particular, the Office completely ignores specific recitation of targets for the recited first and second multi-way compare and swaps. Second, the Office completely ignores specific recitation of use of the recited first and second multi-way compare and swaps to disambiguate certain recited retry and boundary condition states. It is as if the Office views such limitations as having no patentable significance. Failure to consider each limitation of the claims is, of course, legal error.

As with the previously analyzed claims, none of relied upon prior art discloses or suggests the specifically-recited limitations. Indeed, *Arnold's* methods (and code) cannot reasonably be construed as employing a multi-way compare and swap to disambiguate retry and boundary condition states since Arnold's only post-CDS decision is to retry on failure and exit on success.

For at least the foregoing reasons, independent claim 10 and those dependent therefrom are all allowable over the art of record.

Dependent Claim 12

Arnold fails to disclose or suggest, taken alone or in combination, *full state* detection in an array. As before, absence of such disclosure is not surprising since *Arnold* concerns manipulation of list data structures (unbounded in size), not a double-ended data structure of bounded size (as recited in the language of claim 10, from which claim 12 depends). *Greenspan* does not alter the conclusion. Accordingly, claim 12 is allowable for this reason as well.

Claims 15, 20 and 23

Turning to claims 15, 20 and 23, Applicant's claims once again recite specific targets for the atomic dual-target compare and swap operations employed. As before, nothing in *Arnold* (or *Greenspan*) discloses or suggests, taken alone or in combination, the use of respective DCASs operating on the recited targets, i.e.,

- a first end index and a deque element adjacent to that identified by the first end index (pop operation, independent claim 15),
- a first end index and a deque element adjacent to that identified by the first end index (pop operation) and a third end index and a deque element identified by the third end index (push operation, dependent claim 20), and
- a first end index and a deque element identified by the first end index (push operation, independent claim 23).

Furthermore, both *Arnold* and *Greenspan* are devoid of disclosure to the effect that such DCASs are used to interrogate instantaneous values of the recited targets for respective signatures indicative of:

- an empty state (pop operation, independent claim 15),
- an empty state (pop operation) and a full state (push operation, dependent claim 20), and
- a full state (push operation, independent claim 23) and

For at least the foregoing reasons, claims 15, 20 and 23, together with those dependent therefrom, are all allowable over the art of record. For completeness, we note that the Office relies, in addition, on *MacGregor* in its rejection of dependent claims 17-19. However,

consistent with the reasoning above, *MacGregor* does not disclose or suggest the missing limitations and claims 17-19 are similarly allowable.

CONTENTIONS WITH RESPECT TO GROUND III

Claims 32, 35, 40 and 42 have been finally rejected under 35 U.S.C. §103(a) as being unpatentable over *MacGregor* in view of *Greenspan*, and further in view of Mark Allen Weiss's Data Structures and Algorithm Analysis in C++ Second Edition © 1999 (hereinafter "*Weiss*"). Claims 32 and 40 are independent. *MacGregor* and *Greenspan* are relied upon as previously described, while the Office turns to *Weiss* for the missing disclosure of a circular buffer.

Independent Claim 32

The scope of claim 32 differs substantially from those discussed above. Nonetheless, several errors in the rejection of claim 32 (and those dependent therefrom) mimic those described above. In particular, claim recites a double-ended queue (deque) implementation including:

1) an array (more particularly, a **contiguous array S of bounded size**);

2) computer readable encodings of an access operation that employs atomic dual target compare and swaps (DCAS) to update a pair of specific recited targets. The targets include either the L or a R index of into the contiguous array AND an **element of the contiguous array** corresponding to the respective indices.

As before, neither *MacGregor* nor *Greenspan* discloses or suggest these limitations of invention(s) as claimed. While *Weiss* discloses a circular buffer implementation of an array, it is notable that *Weiss'* implementation is a textbook implementation that makes no provision for concurrent access. As a result, not only are DCAS-based access operations not taught, but mechanisms for synchronization of concurrent accesses are contraindicated.

At best, *Weiss* suggests that if one had developed a solution for concurrent access to an array (which neither *MacGregor* nor *Greenspan* actually teach), one underlying coding scheme *to try in developing solutions* would be a circular buffer. However, *Weiss* neither discloses nor suggests a concurrent array-based solution, nor provides guidance as to how any alleged teaching of *MacGregor* and *Greenspan* could be modified to employ atomic dual target compare and swaps as claimed. Rather, even *assuming arguendo* disclosure not present in *MacGregor* or

Greenspan, the Office's rejection of claim 32 necessarily engages in impermissible hindsight using applicant's claim as a template.

In making an assessment of differences between the prior art and the claimed subject matter, section 103 specifically requires consideration of the claimed invention "as a whole." Ruiz v. A.B. Chance Co., 357 F.3d 1270, 1275, 69 U.S.P.Q.2d (BNA) 1686, 1690 (Fed. Cir. 2004). Inventions typically are new combinations of existing principles or features. Envtl. Designs, Ltd. v. Union Oil Co., 713 F.2d 693, 698, 218 U.S.P.Q. (BNA) 865 (Fed. Cir. 1983) (noting that "virtually all [inventions] are combinations of old elements"). The "as a whole" instruction in title 35 prevents evaluation of the invention part by part. Ruiz, 357 F.3d at 1275, 69 U.S.P.Q.2d at 1690. Without this important requirement, an obviousness assessment might successfully break an invention into its component parts, then find a prior art reference corresponding to each component or (as here) unrelated features of a single reference. This line of reasoning would import hindsight into the obviousness determination by using the invention as a roadmap to find its prior art components. Further, this improper method would discount the value of combining various existing features or principles in a new way to achieve a new result—often the essence of invention. Id.

Accordingly, claim 32 and those dependent therefrom are all allowable for at least these reasons and a notice to that effect is respectfully requested.

Independent Claim 40

The scope of claim 40 differs substantially from that of claim 32. Nonetheless, several errors in the rejection of claim 40 (and those dependent therefrom) mimic those described above. In particular, claim recites a concurrent shared object implementation including:

1) an object instantiable as a circular buffer of bounded size implementing a **contiguous array**;

2) instances of at least one functional sequence concurrently executable by plural processors of a multiprocessor, each instance including an atomic dual target compare and swap (DCAS) to update respective pairs of targets. The targets include an end-identifying index AND **an element of the array** corresponding to a then-current value of the index; and

3) **the DCAS of at least one the functional sequences being responsive to a boundary condition state.**

As before, neither *MacGregor* nor *Greenspan* discloses or suggest these limitations of invention(s) as claimed. Neither *MacGregor* nor *Greenspan* discloses or suggests a DCAS that is responsive to a boundary condition state. While *Weiss* discloses a circular buffer implementation of an array, it is notable that *Weiss'* implementation is a textbook implementation that makes no provision for concurrent access. As a result, not only are DCAS-based access operations not taught, but mechanisms for synchronization of concurrent accesses are contraindicated.

Accordingly, claim 40 and those dependent therefrom are all allowable for at least these reasons and a notice to that effect is respectfully requested.

CONTENTIONS WITH RESPECT TO GROUND IV

Claims 33-34 and 41 have been finally rejected under 35 U.S.C. §103(a) as being unpatentable over *MacGregor* in view of *Greenspan* and in view of *Weiss* as applied to claim 32, and further in view of *Arnold.*

Dependents Claim 33, 34 and 41

As before *MacGregor*, nor *Greenspan* nor *Weiss* disclose or suggest, taken alone or in combination, *full state* or *empty state* detection in an array using an indication returned from a DCAS. *Arnold* does not alter this conclusion. Furthermore, with respect to the *full state* detection mechanisms recited in claims 33 and 41, absence of such disclosure is not surprising since neither *MacGregor* nor *Arnold* concerns manipulation of list data structures (unbounded in size), not a contiguous array of bounded size (as recited in the language of claim 32, from which claim 33 depends) or a circular buffer of bounded size (as recited in the language of claim 40, from which claim 41 depends). Claims 33, 34 and 41 are allowable for at least these reasons as well.

CONTENTIONS WITH RESPECT TO GROUND V

Claim 43 has been finally rejected under 35 U.S.C. §103(a) as being unpatentable over *Arnold* in view of *Greenspan*, and further in view of *Weiss*. Although, claim 43 differs substantially in scope, this rejection cannot stand for the same reasons as those discussed in the contention with respect to Ground II rejections. In particular, claim 43 recites "the coordinating means employing...at least one atomic dual target compare and swap (DCAS) operation to disambiguate a retry state and a boundary condition state of the array...."

Despite creative interpretations of the relied upon references, there is simply no disclosure or suggestion for employing an atomic dual target compare and swap (DCAS) operation to disambiguate a retry state and a boundary condition state. *Arnold* uses a BC instruction to check for a retry state (*see* lines 5 and 16 of Table 2 at col. 8) and an LTR instruction to check for an empty queue (*see* line 2 of Table 2 at col. 8) and, as previously explained, *Arnold's* implementation is simply not concerned with *full states*. Indeed, *Arnold's* methods (and code) cannot reasonably be construed as employing an atomic dual target compare and swap to disambiguate retry and boundary condition states since Arnold's only post-CDS decision is to retry on failure and exit on success.

None of the art of record, standing alone or in combination, discloses or suggests each and every limitation of claim 43. Hence, claim 43 is nonobvious and allowable.

CONCLUSION

For the at least the foregoing reasons, Appellants' presently claimed invention would not have been obvious to one of ordinary skill in the art under 35 U.S.C. § 103(a) in view of the cited prior art. Accordingly, this honorable Board is respectfully requested to reverse the rejections of claims 1 – 43 and to direct the claims of the present application to be issued.

CERTIFICATE OF MAILING OR TRANSMISSION

I hereby certify that, on the date shown below, this correspondence is being

☒ deposited with the US Postal Service with sufficient postage as first class mail and addressed as shown above.

☐ facsimile transmitted to the US Patent and Trademark Office.

David W. O'Brien — Date: 7-Nov-05

EXPRESS MAIL LABEL: _____

Respectfully submitted,

David W. O'Brien, Reg. No. 40,107
Attorney for Applicant(s)
(512) 338-6314 (direct)
(512) 338-6300 (main)
(512) 338-6301 (fax)

CLAIMS APPENDIX

1. A method of managing access to an array susceptible to concurrent operations on a sequence encoded therein, the method comprising:

executing as part of a pop operation, an atomic dual target compare and swap (DCAS) to update a then-current, end identifying index for the array and an element of the array adjacent to that identified by the end identifying index; and

returning from the DCAS, on failure thereof, an indication by which an empty state of the array is detectable.

2. The method of claim 1,

wherein the indication by which the empty state of the array is detectable is indicative of presence of a distinguishing value in the adjacent element.

3. The method of claim 1, wherein the array encodes a double-ended queue as a circular buffer of bounded size, the end identifying index and an opposing end identifying index delimiting the sequence.

4. The method of claim 1,

wherein the pop operation is a left pop operation;

wherein the end identifying index is a left-end index; and

wherein the adjacent element is to the right of the identified element.

5. The method of claim 1,

wherein the pop operation is a right pop operation;

wherein the end identifying index is a right-end index; and

wherein the adjacent element is to the left of the identified element.

6. A method of managing access to an array susceptible to concurrent operations on a sequence encoded therein, the method comprising:

executing as part of a push operation, an atomic dual target compare and swap (DCAS) to update a then-current, end identifying index for the array and an element of the array identified by the end identifying index; and

returning from the DCAS, on failure thereof, an indication by which a full state of the array is detectable.

7. The method of claim 6,

wherein the indication by which the full state of the array is detectable is indicative of absence of a distinguishing value in the identified element.

8. The method of claim 6,

wherein the push operation is a left push operation; and

wherein the end identifying index is a left-end index.

9. The method of claim 6,

wherein the pop operation is a right push operation; and

wherein the end identifying index is a right-end index.

10. A method of providing concurrent access to a double-ended data structure of bounded size implemented using a circular buffer technique, the method comprising:

as part of an access to a first-end of the double-ended data structure, performing in alternate legs of a conditional branch:

a first atomic multi-way compare and swap on then-current contents of a first-end index store and a corresponding element of the double-ended data structure to disambiguate a retry state and a boundary condition state of the double-ended data structure;

a second atomic multi-way compare and swap on then-current contents of the first-end index store and a corresponding element of the double-ended data structure, the second multi-way compare and swap performing the access and, on failure thereof, returning an indication disambiguating a retry state and the boundary condition state of the double-ended data structure,

wherein the conditional branch discriminates between presence and absence of a distinguishing value in an element of the double-ended data structure corresponding to the then-current contents of the first-end index store.

11. The method of claim 10,

wherein the access includes a pop from the first-end of the double-ended data structure;

wherein the boundary condition state is an empty state of the double-ended data structure; and

wherein the retry state results from a concurrently performed push or pop access at the first-end of the double-ended data structure.

12. The method of claim 10,

wherein the access includes a push onto the first-end of the double-ended data structure;

wherein the boundary condition state is a full state of the double-ended data structure; and

wherein the retry state results from a concurrently performed push or pop access at the first-end of the double-ended data structure.

13. The method of claim 10, wherein the double-ended data structure includes a double-ended queue (deque).

14. The method of claim 10, wherein the multi-way compare and swap is a dual target compare and swap (DCAS).

15. A method of managing concurrent access to double-ended queue (deque), the method comprising:

employing, in an implementation of a pop operation, execution of a an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a first end index and a deque element adjacent to that identified thereby for a signature indicative of an empty state of the array, the signature including presence in that adjacent element of a distinguishing value,

wherein successful execution of an opposing end pop operation includes execution of a DCAS to update a second end index and a deque element adjacent to that identified thereby, the update of that adjacent element storing the distinguishing value therein.

16. The method of claim 15, further comprising:

wherein successful execution of a competing, same end pop operation includes execution of a DCAS to update the first end index and a deque element adjacent to that identified thereby, the update of that adjacent element storing the distinguishing value therein.

17. The method of claim 15, further comprising:

wherein the first end index is a left index and, if the state of the deque is non-empty, the deque element adjacent to that identified thereby is a left most element of the deque;

wherein the second end index is a right index and, if the state of the deque is non-empty, the deque element adjacent to that identified thereby is the right most element of the deque.

18. The method of claim 15,

wherein the pop operation is a left pop operation and the opposing end pop operation is a right pop operation; and

wherein the first end index is a left end index and the element adjacent to that identified thereby is adjacent to the right.

19. The method of claim 15, wherein the distinguishing value is encoded as a null value.

20. The method of claim 15, further comprising:

employing, in an implementation of a push operation, execution of an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a third end index and a deque element identified thereby for a signature indicative of an full

state of the deque, the signature including absence in that identified deque element of a distinguishing value,

wherein successful execution of an opposing end push operation includes execution of a DCAS to update a fourth end index and a deque element identified thereby, the update of the identified deque element storing a value other than the distinguishing value therein.

21. The method of claim 20,

wherein the first end index and the third end index identify a same end of the deque; and

wherein the second end index and the fourth end index identify a same end of the deque.

22. The method of claim 20,

wherein the first end index and the fourth end index identify a same end of the deque; and

wherein the second end index and the third end index identify a same end of the deque.

23. A method of managing concurrent access to double-ended queue (deque), the method comprising:

employing, in an implementation of a push operation, execution of a an atomic dual target compare and swap (DCAS) to interrogate instantaneous values of a first end index and a deque element identified thereby for a signature indicative of a full state of the deque, the signature including absence in that identified deque element of a distinguishing value,

wherein successful execution of an opposing end push operation includes execution of a DCAS to update an opposing end index and a deque element identified thereby, the update of the identified deque element storing a value other than the distinguishing value therein.

24. The method of claim 23, further comprising:

wherein successful execution of a competing, same end push operation includes execution of a DCAS to atomically update the first end index and a deque element identified thereby, the update of that adjacent element storing a value other than the distinguishing value therein.

25. A method of managing concurrent access to an array susceptible to competing accesses at same and opposing ends thereof, the method comprising:

executing as part of a first access operation, an atomic dual target compare and swap (DCAS) to update a first end identifying index and an element of the array corresponding to a then-current value thereof;

executing as part of a competing second access operation, a DCAS to update a second end identifying index and an element of the array corresponding to a then-current value thereof,

wherein, if successful completion of one of the first and the second competing access operations results in a boundary condition state of the array, the DCAS of the other of the first and the second access operations fails and returns an indication thereof.

26. The method of claim 25,

wherein the first access operation and the competing second access operation are competing pop operations;

wherein the array elements corresponding to the first and second indices are each adjacent to that identified by the respective index;

wherein the boundary condition state is an empty state; and

wherein the adjacent element referenced by the failing one of the competing pop operations encodes a distinguishing value signifying the empty state.

27. The method of claim 25,

wherein the competing pop operations are competing opposing end pop operations; and

wherein the first index and the second index identify opposing ends of the array;

28. The method of claim 25,

wherein the competing pop operations are competing same end pop operations; and

wherein the first index and the second index identify a same end of the array;

29. The method of claim 25,

wherein the first access operation and the competing second access operation are competing push operations;

wherein the array elements corresponding to the first and second indices are each identified by the respective index;

wherein the boundary condition state is an full state; and

wherein the array element referenced by the failing one of the competing push operations encodes a value other than a distinguishing value.

30. The method of claim 25,

wherein the competing push operations are competing opposing end push operations; and

wherein the first index and the second index identify opposing ends of the array;

31. The method of claim 25,

wherein the competing push operations are competing same end push operations; and

wherein the first index and the second index identify a same end of the array;

32. A double-ended queue (deque) implementation comprising:

a contiguous array S of bounded size encoded in an addressable store;

a left index L and a right index R into the contiguous array, the contiguous array S, the left index L and the right index R together defining a circular buffer with state including a sequence of zero or more values encoded in the contiguous array between elements S[L] and S[R] thereof and wherein at least the S[L] and S[R] entries encode a distinguishing value;

a computer readable encoding of at least a first access operation, execution of the first access operation operating at a particular end of the sequence and employing an atomic dual target compare and swap (DCAS) to update a corresponding one, but not both, of the left and right indices L and R and an element of the contiguous array adjacent to the contiguous array element identified thereby.

33. The double-ended queue (deque) implementation of claim 32,

wherein the first access operation includes a push; and

wherein, on failure, the DCAS returns an indication by which a full state of the contiguous array is detected.

34. The double-ended queue (deque) implementation of claim 32,
wherein the first access operation includes a pop; and
wherein, on failure, the DCAS returns an indication by which an empty state of the contiguous array is detected.

35. The double-ended queue (deque) implementation of claim 32, further comprising:
computer readable encodings of at least three additional access operations,
wherein the first and three additional access operations include push and pop operations at left and rights end of the sequence, respectively.

36. A concurrent shared object implementation comprising:
a contiguous array encoded in an addressable store;
opposing indices into the contiguous array usable to delimit therebetween a portion of the contiguous array for storage of a sequence of zero or more data values; and
a computer readable encoding of push and pop operations defined to operate on elements of the contiguous array and on respective of the opposing indices,
wherein the push operation employs a first instance of an atomic dual target compare and swap (DCAS) operation to update one of the opposing indices and a corresponding element of the contiguous array while returning on failure, an indication by which a full state of the contiguous array is detected, and
wherein the pop operation employs a second instance of a DCAS operation to update one of the opposing indices and a corresponding element of the contiguous array while returning on failure, an indication by which an empty state of the contiguous array is detected.

37. The concurrent shared object implementation of claim 36,
wherein concurrent shared object includes a deque; and
wherein the computer readable encoding of push and pop operations includes:
opposing end variants of the pop operation; and

opposing end variants of the push operation.

38. The concurrent shared object implementation of claim 36,
wherein concurrent shared object includes a queue or FIFO; and
wherein the computer readable encoding of push and pop operations operate on opposing ends of the queue or FIFO.

39. The concurrent shared object implementation of claim 36,
wherein concurrent shared object includes a stack or LIFO; and
wherein the computer readable encoding of push and pop operations operate on a same end of the stack or LIFO.

40. A computer program product encoded in at least one computer readable medium, the computer program product comprising:
at least one functional sequence implementing an access operation on a concurrent shared object, the concurrent shared object instantiable as a circular buffer of bounded size implementing a contiguous array delimited by a pair of end identifying indices;
instances of the at least one functional sequence concurrently executable by plural processors of a multiprocessor and each including an atomic dual target compare and swap (DCAS) to update a corresponding one of the end identifying indices and an element of the array corresponding to a then-current value thereof; and
the DCAS of the at least one functional sequence responsive to a corresponding boundary condition state of the concurrent shared object.

41. A computer program product as recited in 40,
wherein the at least one functional sequence includes opposing end variants of push and pop operations on the concurrent shared object;
wherein the boundary condition state corresponding to push operations is a full state of the array; and
wherein the boundary condition state corresponding to pop operations is an empty state of the array.

42. A computer program product as recited in 40,

wherein the at least one computer readable medium is selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

43. An apparatus comprising:

plural processors;

a store addressable by each of the plural processors;

first- and second-end index stores accessible to each of the plural processors for identifying opposing ends of a bounded-size contiguous array encoded in circular buffer form in the addressable store; and

means for coordinating competing access operations, the coordinating means employing in each instance thereof, at least one atomic dual target compare and swap (DCAS) operation to disambiguate a retry state and a boundary condition state of the array based on then-current contents of one, but not both, of first- and second-end index stores and an array element corresponding thereto.

EVIDENCE APPENDIX

There is no evidence submitted pursuant to 37 C.F.R. § 1.130, 1.131, or 1.132 or any other evidence entered by the examiner and relied upon by appellant in the appeal.

RELATED APPEALS APPENDIX

There are no decisions rendered by a court or the Board in any proceeding identified above in the Related Appeals and Interferences section.